

In [1]:

```
%%html
<style>
table {float: left}
</style>
```

# Semesterarbeit Statistische Datenanalyse

## Dokumenteninformationen

Titel	todo
Schule	Fernfachhochschule Schweiz
Studiengang	Certificate of Advanced Studies in Statistische Datenanalyse & Datenvisualisierung
Kennung	DS-C-SD001.StatDa.ZH-Sa-1.PVA.FS23
Semester	Frühlingssemester 2023
Dozent	<b>Jörg Osterrieder</b> joerg.osterrieder@ffhs.ch
Autor	<b>Patrick Hirschi</b> Geburtsdatum: 12.01.1990 Matrikelnummer: 10-179-026 Studierenden-ID: 200768 patrick-hirschi@gmx.ch

## Einleitung

Die vorliegende Semesterarbeit hat zum Ziel, auf einem ausgewählten Datenset verschiedene statistische Methoden/Verfahren zu testen, zu interpretieren und zu beurteilen.

## Datenbeschaffung

### Datenset finden

Um böse Überraschungen im Projektverlauf zu vermeiden, wurde an dieser Stelle darauf verzichtet, ein Datenset aus dem eigenen geschäftlichen Umfeld anzufragen. Die Zeit zwischen Semesterstart und Abgabe der Arbeiten ist dafür leider jeweils sehr knapp bemessen.

[Kaggle](#) ist eine Online Community für Data Scientisten, mit vielen offenen Datensets, Code-Challenges und Forenbeiträgen. Um ein Dataset herunterzuladen braucht man lediglich einen Account. Dieser ist kostenlos.

Für diese Arbeit wurde ein [Dataset](#) von der Weltgesundheitsorganisation (WHO) ausgewählt. Darin sind für 193 Länder weltweit und für den Zeitraum 2000-2015 gesundheitsbezogene Daten wie z.B. die Lebenserwartung, Sterberate für Erwachsene, etc. abgebildet. Das Dataset hat 64'636 Zeilen und 22 Spalten (davon die meisten numerisch). Damit ist es sehr gut geeignet für die verschiedenen statistischen Tests, die im Rahmen dieser Arbeit durchgeführt und evaluiert werden sollen.

## Quelldaten laden

Nach dem Herunterladen der Daten in den lokalen Projektordner, sollen sie zuerst in ein Pandas DataFrame eingelesen werden. Dazu müssen aber die relevanten python Module in Jupyter importiert werden.

### Modulimport

```
In [2]: # import required modules
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import sys
import os
import datetime
import requests
from requests.exceptions import HTTPError
import pandas as pd
import numpy as np
import seaborn as sns
import statsmodels.api as sm
from statsmodels.stats.diagnostic import normal_ad
from statsmodels.graphics.gofplots import qqplot
import sklearn as sk
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Visualization
import plotly
import plotly.express as px
import plotly.graph_objects as go

# Scipy
import scipy as sp
from scipy.stats import shapiro
from scipy.stats import kstest

# print versions
```

```

print('matplotlib: %s' % matplotlib.__version__)
print('requests: %s' % requests.__version__)
print('pandas: %s' % pd.__version__)
print('numpy: %s' % np.__version__)
print('scipy: %s' % sp.__version__)
print('sklearn: %s' % sk.__version__)
print('plotly: %s' % plotly.__version__)
print('seaborn: %s' % sns.__version__)
print('statsmodels: %s' % sm.__version__)

# log success
print(f'The module import was successful!')

```

```

matplotlib: 3.6.3
requests: 2.28.2
pandas: 1.5.3
numpy: 1.24.2
scipy: 1.10.1
sklearn: 1.2.1
plotly: 5.13.0
seaborn: 0.12.2
statsmodels: 0.13.5
The module import was successful!

```

## Datenimport

```

In [3]: # datetime method for logging purposes
def get_current_time_str():
    return datetime.datetime.now().strftime("%H:%M:%S.%f")

# assign directory
directory = './data/'

# read input data
df_input_data = pd.read_csv(os.path.join(directory, 'life_expectancy_data.csv'))

# log an output message
print(f'{get_current_time_str()}: Successfully loaded input data '
      f'of size: {df_input_data.size}')

```

23:10:56.149565: Successfully loaded input data of size: 64636

```

In [4]: # print the first 10 rows to check if the import worked
df_input_data.head(10)

```

Out[4]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hep.
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	
5	Afghanistan	2010	Developing	58.8	279.0	74	0.01	79.679367	
6	Afghanistan	2009	Developing	58.6	281.0	77	0.01	56.762217	
7	Afghanistan	2008	Developing	58.1	287.0	80	0.03	25.873925	
8	Afghanistan	2007	Developing	57.5	295.0	82	0.02	10.910156	
9	Afghanistan	2006	Developing	57.3	295.0	84	0.03	17.171518	

10 rows × 22 columns

In [5]:

```
# check the column types
df_input_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Country          2938 non-null   object 
 1   Year              2938 non-null   int64  
 2   Status             2938 non-null   object 
 3   Life expectancy    2928 non-null   float64
 4   Adult Mortality    2928 non-null   float64
 5   infant deaths     2938 non-null   int64  
 6   Alcohol            2744 non-null   float64
 7   percentage expenditure  2938 non-null   float64
 8   Hepatitis B        2385 non-null   float64
 9   Measles            2938 non-null   int64  
 10  BMI                2904 non-null   float64
 11  under-five deaths  2938 non-null   int64  
 12  Polio               2919 non-null   float64
 13  Total expenditure   2712 non-null   float64
 14  Diphtheria          2919 non-null   float64
 15  HIV/AIDS            2938 non-null   float64
 16  GDP                 2490 non-null   float64
 17  Population          2286 non-null   float64
 18  thinness 1-19 years  2904 non-null   float64
 19  thinness 5-9 years   2904 non-null   float64
 20  Income composition of resources 2771 non-null   float64
 21  Schooling           2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB
```

```
In [6]: # rename the columns to remove spaces and shorten them
df_input_data.rename(columns={'Country': 'country',
                             'Year': 'year',
                             'Status': 'status',
                             'Life expectancy ': 'life_exp',
                             'Adult Mortality': 'adlt_mort',
                             'infant deaths': 'inft_death',
                             'Alcohol': 'alcohol',
                             'percentage expenditure': 'perc_exp',
                             'Hepatitis B': 'hep_b',
                             'Measles ': 'measles',
                             ' BMI ': 'bmi',
                             'under-five deaths ': 'u5_death',
                             'Polio': 'polio',
                             'Total expenditure': 'total_exp',
                             'Diphtheria ': 'dipht',
                             ' HIV/AIDS': 'hiv',
                             'GDP': 'gdp',
                             'Population': 'population',
                             ' thinness 1-19 years': 'thinness_teenager',
                             ' thinness 5-9 years': 'thinness_children',
                             'Income composition of resources': 'incm_comp'
                             'Schooling': 'schooling'},
                           inplace=True)

# check the column names
df_input_data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   country          2938 non-null    object  
 1   year              2938 non-null    int64  
 2   status             2938 non-null    object  
 3   life_exp           2928 non-null    float64 
 4   adlt_mort          2928 non-null    float64 
 5   inft_death         2938 non-null    int64  
 6   alcohol            2744 non-null    float64 
 7   perc_exp           2938 non-null    float64 
 8   hep_b              2385 non-null    float64 
 9   measles            2938 non-null    int64  
 10  bmi                2904 non-null    float64 
 11  u5_death           2938 non-null    int64  
 12  polio              2919 non-null    float64 
 13  total_exp          2712 non-null    float64 
 14  dipt               2919 non-null    float64 
 15  hiv                2938 non-null    float64 
 16  gdp                2490 non-null    float64 
 17  population          2286 non-null    float64 
 18  thinness_teenager   2904 non-null    float64 
 19  thinness_children    2904 non-null    float64 
 20  incm_comp           2771 non-null    float64 
 21  schooling            2775 non-null    float64 
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB

```

Die Spaltenbeschreibungen gibt es auf Kaggle nur in Englisch. Der Vollständigkeit halber wurden sie hier auf deutsch übersetzt und zusammengetragen:

Spaltenname (neu)	Spaltenname (alt)	Beschreibung
country	Country	Land
year	Year	Jahr
status	Status	Entwicklungsstatus - "Developed" (für entwickelte Staaten) oder "Developing" (für Entwicklungsländer)
life_exp	Life expectancy	Lebenserwartung (Alter in Jahren)
adlt_mort	Adult Mortality	Erwachsenensterblichkeit (Anzahl Tote auf 1000 Einwohner für die Altersklasse 15 bis 60)
inft_death	infant deaths	Kindersterblichkeit (Anzahl Tote auf 1000 Einwohner für die Altersklasse 0 bis 15)
alcohol	Alcohol	Alkohol in Liter pro Kopf und Jahr
perc_exp	percentage expenditure	Anteil der Gesundheitsausgaben am BIP pro Kopf (in Prozent)
hep_b	Hepatitis B	Hepatitis B Immunisierung bei 1 Jahr alten Kindern (in Prozent)
measles	Measles	Anzahl gemeldete Masern-Fälle pro 1000 Einwohner

Spaltenname (neu)	Spaltenname (alt)	Beschreibung
bmi	BMI	Durchschnittlicher Body Mass Index
u5_death	under-five deaths	Anzahl Tote unter 5 Jahren pro 1000 Einwohner
polio	Polio	Polio Immunisierung bei 1 Jahr alten Kindern (in Prozent)
total_exp	Total expenditure	Anteil der Gesundheitsausgaben an den Gesamtausgaben (in Prozent)
diph	Diphtheria	Diphtherie Immunisierung bei 1 Jahr alten Kindern (in Prozent)
hiv	HIV/AIDS	HIV/AIDS Tote pro 1000 Lebendgeburten (0-4 Jahre)
gdp	GDP	BIP pro Kopf (in US Dollar)
population	Population	Bevölkerungsanzahl
thinness_teenager	thinness 10-19 years	Prävalenz von Untergewicht bei Kindern und Jugendlichen im Alter von 10 bis 19 Jahren (in %)
thinness_children	thinness 5-9 years	Prävalenz von Untergewicht bei Kindern und Jugendlichen im Alter von 5 bis 9 Jahren (in %)
incm_comp	Income composition of resources	Index der menschlichen Entwicklung in Bezug auf die Einkommenszusammensetzung der Ressourcen (Index von 0 bis 1)
schooling	Schooling	Anzahl der Schuljahre pro Kopf

## Datenanalyse

### Explorative Datenanalyse

Mit der explorativen Datenanalyse sollen die Daten besser verstanden werden können. Es geht um die Anwendung grundlegender statistischer und visueller Analysen. So können Probleme (z.B. fehlende Werte), Verteilungen oder Wertebereiche bereits früh im Projekt erkannt werden.

### Generelle Prüfungen

Die Pandas Info-Methode gibt einen schnellen Überblick über das Dataframe.

```
In [7]: # Check data for datatypes, column/row counts, missing elements, etc.
df_input_data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   country          2938 non-null    object  
 1   year              2938 non-null    int64  
 2   status             2938 non-null    object  
 3   life_exp           2928 non-null    float64 
 4   adlt_mort          2928 non-null    float64 
 5   inft_death         2938 non-null    int64  
 6   alcohol            2744 non-null    float64 
 7   perc_exp           2938 non-null    float64 
 8   hep_b              2385 non-null    float64 
 9   measles            2938 non-null    int64  
 10  bmi                2904 non-null    float64 
 11  u5_death           2938 non-null    int64  
 12  polio              2919 non-null    float64 
 13  total_exp          2712 non-null    float64 
 14  dipt               2919 non-null    float64 
 15  hiv                2938 non-null    float64 
 16  gdp                2490 non-null    float64 
 17  population          2286 non-null    float64 
 18  thinness_teenager   2904 non-null    float64 
 19  thinness_children    2904 non-null    float64 
 20  incm_comp           2771 non-null    float64 
 21  schooling            2775 non-null    float64 
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB

```

Es gibt also 22 Spalten und 2938 Zeilen. Alle Spalten sind numerisch, ausser die Spalten Land und Status. Es ist auch ersichtlich, dass es in einigen Spalten fehlende Werte gibt. Um tiefer in einzelne Spalten einzutauchen, können für jede Spalte die Anzahl Werte, Mittelwert, Standardabweichung, der Minimal-/Maximalwert und die Quartile angezeigt werden.

```
In [8]: # Check count, mean, std, quartiles, min and max per variable
df_input_data.describe()
```

	year	life_exp	adlt_mort	inft_death	alcohol	perc_exp
<b>count</b>	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000
<b>mean</b>	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295
<b>std</b>	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858
<b>min</b>	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000
<b>25%</b>	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343
<b>50%</b>	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906
<b>75%</b>	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144
<b>max</b>	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610

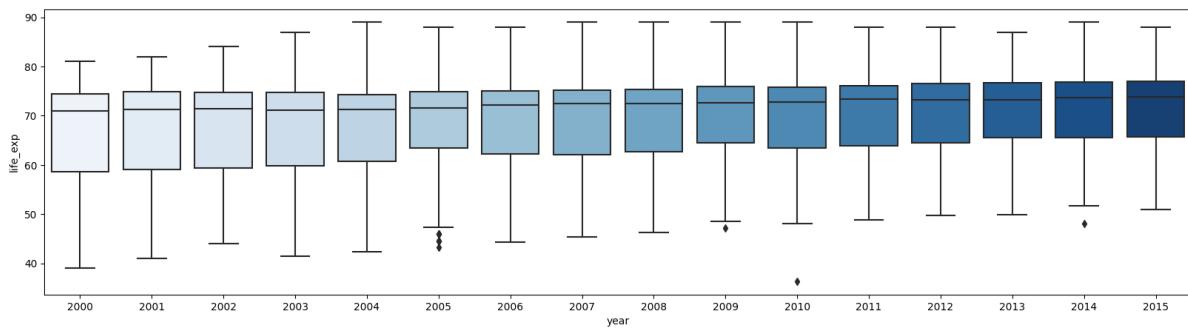
## Univariat

Die Hauptvariable des Datensets ist die Lebenserwartung. Zunächst soll geprüft werden, wie sich die Lebenserwartung über die Jahre verändert hat. Dazu wird ein Boxplot pro Jahr erstellt.

### Boxplot

```
In [9]: fig, ax = plt.subplots(figsize=(20,5))
sns.boxplot(x = df_input_data.year,
             y = df_input_data['life_exp'],
             ax = ax,
             palette="Blues")
```

```
Out[9]: <AxesSubplot: xlabel='year', ylabel='life_exp'>
```



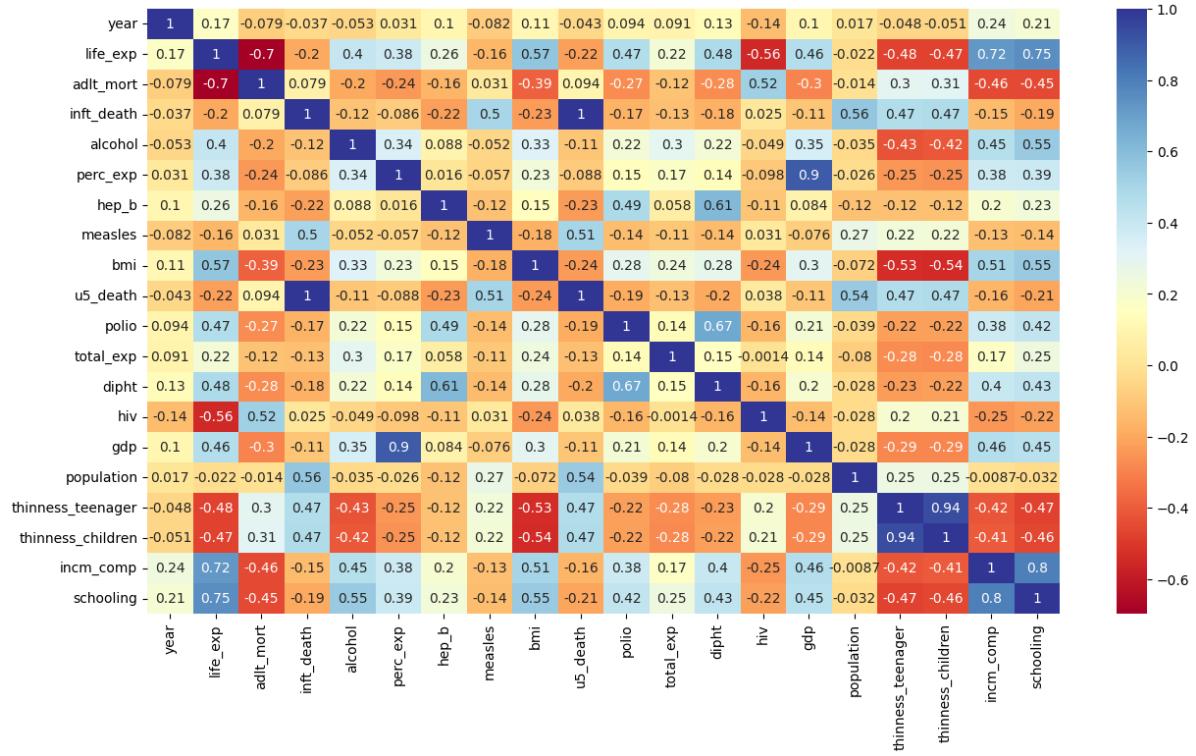
Die mittlere Lebenserwartung (Linie innerhalb der Box) ist von 2000-2015 stets gestiegen. Bei der minimalen Lebenserwartung gibt es aber größere Schwankungen. Beispielsweise ist sie von 2002 auf 2003 plötzlich wieder gesunken. Dies könnte aber auch daher kommen, dass für ein Land die Messung erst ab 2003 durchgeführt wurde.

## Bi-/Multivariat

### Korrelationsmatrix

Die Zahlen zeigen die Korrelation zwischen zwei Variablen. Ein Wert von 0 bedeutet, dass kein Zusammenhang besteht, währenddessen 1 und -1 ein perfekter positiver bzw. negativer linearer Zusammenhang beschreibt. Da immer zwei Variablen miteinander verglichen werden, ist die Diagonale (Variable mit sich selber) immer 1 und der Rest ist an dieser Diagonale gespiegelt (Abhängigkeit von Variablen A und B ist gleich wie von Variablen B und A).

```
In [10]: plt.subplots(figsize=(15,8))
sns.heatmap(df_input_data.corr(numeric_only = True), annot=True, cmap="RdYlE
plt.show()
```



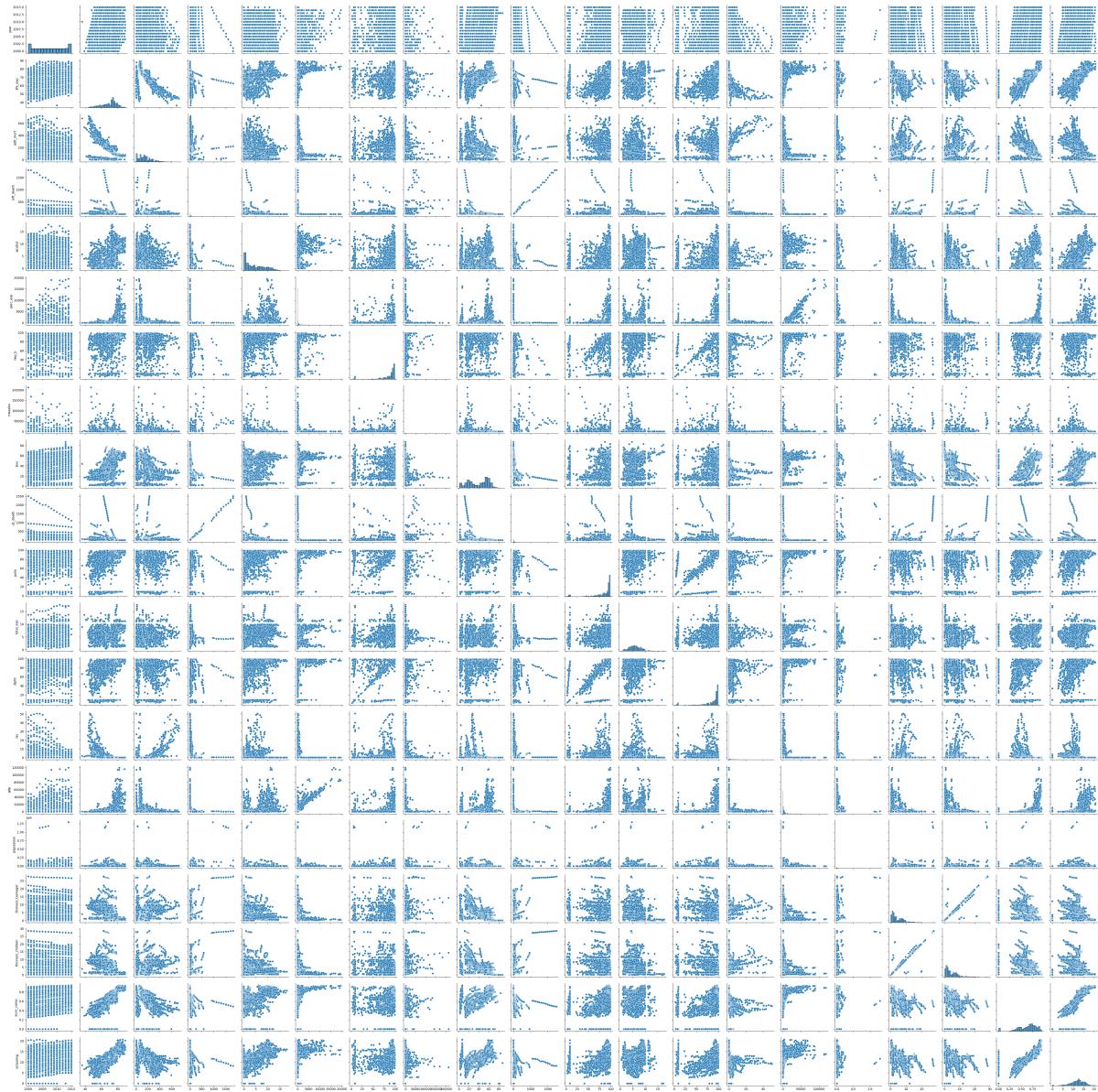
Folgende Erkenntnisse lassen sich aus der Heatmap herauslesen:

- Jede Variable ist mit sich selbst perfekt positiv linear korreliert (nicht sehr überraschend)
- Die Kindersterblichkeit ist mit der Anzahl Toten unter 5 Jahren perfekt positiv linear korreliert (nicht sehr überraschend)
- Es gibt einen starken positiven linearen Zusammenhang zwischen Untergewicht bei Kindern und Untergewicht bei Jugendlichen (nicht sehr überraschend)
- Es gibt einen sehr starken positiven linearen Zusammenhang zwischen dem BIP und dem Anteil der Gesundheitsausgaben am BIP (nicht sehr überraschend)
- Es gibt einen sehr starken positiven linearen Zusammenhang zwischen Entwicklung und Bildung (nicht sehr überraschend)
- Es gibt ein paar moderate positive lineare Zusammenhänge, z.B. zwischen:
  - Polio und Diphtherie
  - Hepatitis B und Diphtherie
  - BMI und Lebenserwartung
  - BIP und Lebenserwartung
  - Masern und Kindersterblichkeit
  - Bildung/Entwicklung und Lebenserwartung
- Es gibt einen starken negativen linearen Zusammenhang zwischen der Erwachsenensterblichkeit und der Lebenserwartung (nicht sehr überraschend)
- Es gibt ein paar moderate negative lineare Zusammenhänge, z.B. zwischen:
  - Untergewicht bei Jugendlichen/Kindern und BMI
  - Untergewicht bei Jugendlichen/Kindern und Lebenserwartung
  - Untergewicht bei Jugendlichen/Kindern und Bildung/Entwicklung
  - HIV und Lebenserwartung

## Pairplot

Mit einem Pairplot kann man wie in der Heatmap jede Kombination von Variablen gegenseitig prüfen. Diesmal wird das Resultat aber grafisch dargestellt. Es sind dieselben Schlussfolgerungen herauszulesen, wie oben beschrieben.

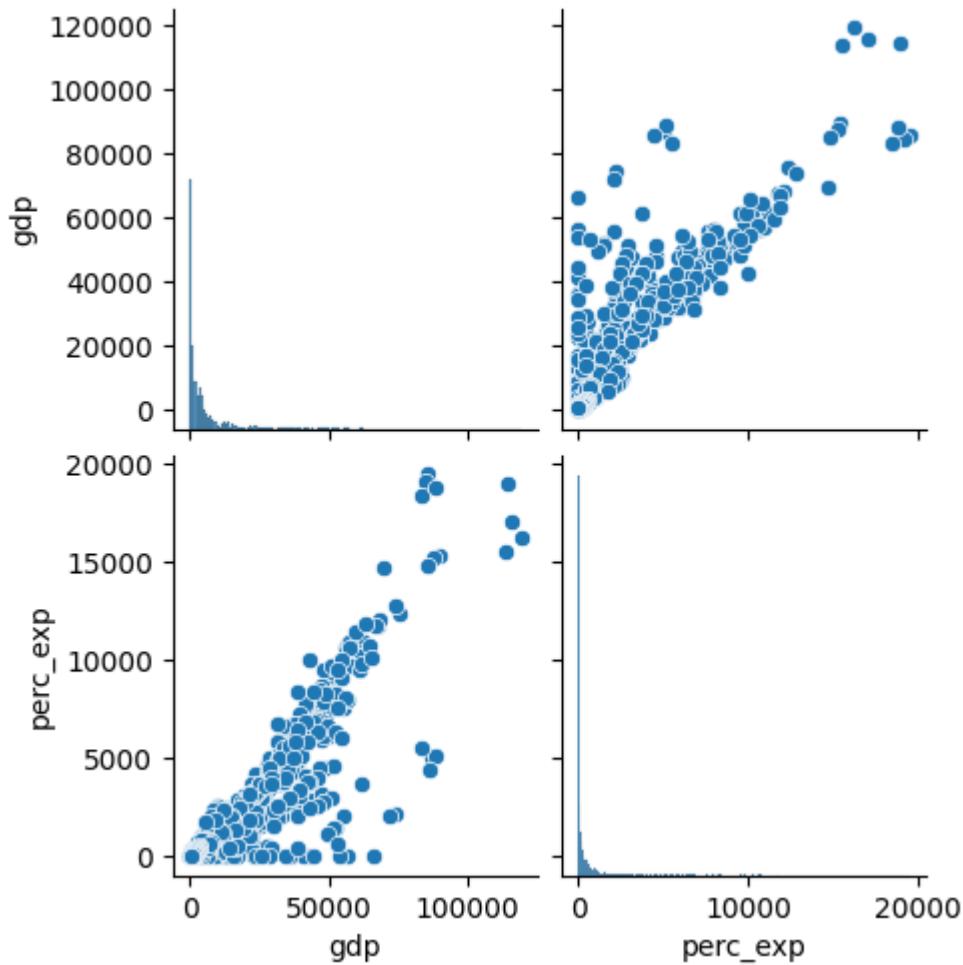
```
In [11]: # seaborn pairplot  
sns.pairplot(df_input_data)  
# to show  
plt.show()
```



Der Zusammenhang zwischen dem BIP und dem Anteil der Gesundheitsausgaben am BIP kann auch isoliert betrachtet werden:

```
In [12]: # seaborn pairplot  
sns.pairplot(df_input_data[['gdp', 'perc_exp']])
```

```
# to show  
plt.show()
```



## Regressionsanalyse

### Multiple Lineare Regression

Die multiple lineare Regression ist eine Methode, um die Werte einer abhängigen Variablen (hier die Lebenserwartung) anhand mehrerer unabhängiger Werte (z.B. sozioökonomische Faktoren, Prävalenzen, etc.) vorherzusagen.

#### Modelldefinition

```
In [13]: x = df_input_data[['population',  
                      'gdp',  
                      'bmi',  
                      'polio',  
                      'diph',  
                      'hiv',  
                      'alcohol',  
                      'measles',  
                      'hep_b',
```

```
'inft_death',
'adlt_mort',
'thinness_teenager',
'thinness_children',
'total_exp']]  
y = df_input_data['life_exp']

x = sm.add_constant(x) # adding a constant

model = sm.OLS(y, x, missing='drop').fit()
predictions = model.predict(x)

print_model = model.summary()
print(print_model)
```

### OLS Regression Results

```
=====
===
Dep. Variable: life_exp R-squared: 0.
747
Model: OLS Adj. R-squared: 0.
745
Method: Least Squares F-statistic: 34
4.3
Date: Sat, 17 Jun 2023 Prob (F-statistic): 0.00
Time: 23:11:30 Log-Likelihood: -479
2.4
No. Observations: 1649 AIC: 96
15.
Df Residuals: 1634 BIC: 96
96.
Df Model: 14
```

Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025
0.975]					
const	64.1298	0.716	89.586	0.000	62.726
65.534					
population	6.11e-09	2.12e-09	2.878	0.004	1.95e-09
1.03e-08					
gdp	0.0001	1.1e-05	11.360	0.000	0.000
0.000					
bmi	0.0863	0.007	12.206	0.000	0.072
0.100					
polio	0.0241	0.006	3.801	0.000	0.012
0.037					
diphth	0.0406	0.007	5.613	0.000	0.026
0.055					
hiv	-0.4536	0.022	-20.545	0.000	-0.497
-0.410					
alcohol	0.2586	0.033	7.744	0.000	0.193
0.324					
measles	1.323e-05	1.3e-05	1.016	0.310	-1.23e-05
3.88e-05					
hep_b	-0.0066	0.005	-1.199	0.231	-0.017
0.004					
inft_death	-0.0047	0.002	-3.059	0.002	-0.008
-0.002					
adlt_mort	-0.0245	0.001	-21.743	0.000	-0.027
-0.022					
thinness_teenager	-0.1715	0.065	-2.627	0.009	-0.300
-0.043					
thinness_children	0.0520	0.065	0.806	0.421	-0.075
0.179					

```

total_exp          0.1230      0.050      2.454      0.014      0.025
               0.221
=====
===
Omnibus:            99.785   Durbin-Watson:        0.
857
Prob(Omnibus):     0.000   Jarque-Bera (JB):    162.
832
Skew:              -0.473   Prob(JB):           4.38e
-36
Kurtosis:           4.214   Cond. No.         4.71e
+08
=====
===

```

#### Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 4.71e+08. This might indicate that there are strong multicollinearity or other numerical problems.

Die multiple lineare Regression zeigt ein Bestimmtheitsmass ( $R^2$ ) von 0.747. Das bedeutet, dass 74.7% der Variabilität der abhängigen Variable durch das lineare Modell erklärt werden kann. Die Konstante (engl. Intercept) des Modelles ist 64.13. Die Koeffizienten für die weiteren Variablen geben den Multiplikator in der Gleichung der linearen Regression an. Das Modell lässt sich also mit der folgenden Gleichung beschreiben:

```
*life_exp = 6.11e-09 * population + 0.0001 * gdp + 0.0863 * bmi + 0.0241 * polio +
0.0406 * dipht - 0.4536 * hiv + 0.2586 * alcohol + 1.323e-05 * measles - 0.0066 *
hep_b - 0.0047 * inft_death - 0.0245 * adlt_mort - 0.1715 * thinness_teenager +
0.0520 * thinness_children + 0.1230 * total_exp + 64.1298*
```

Die p-Werte der einzelnen unabhängigen Variablen beschreiben die Signifikanz. Gehen wir von einem Signifikanzniveau von 0.05 aus, sind also alle Variablen signifikant mit Ausnahme von "measles", "hep\_b" und "thinness\_children" (grösser als 0.05).

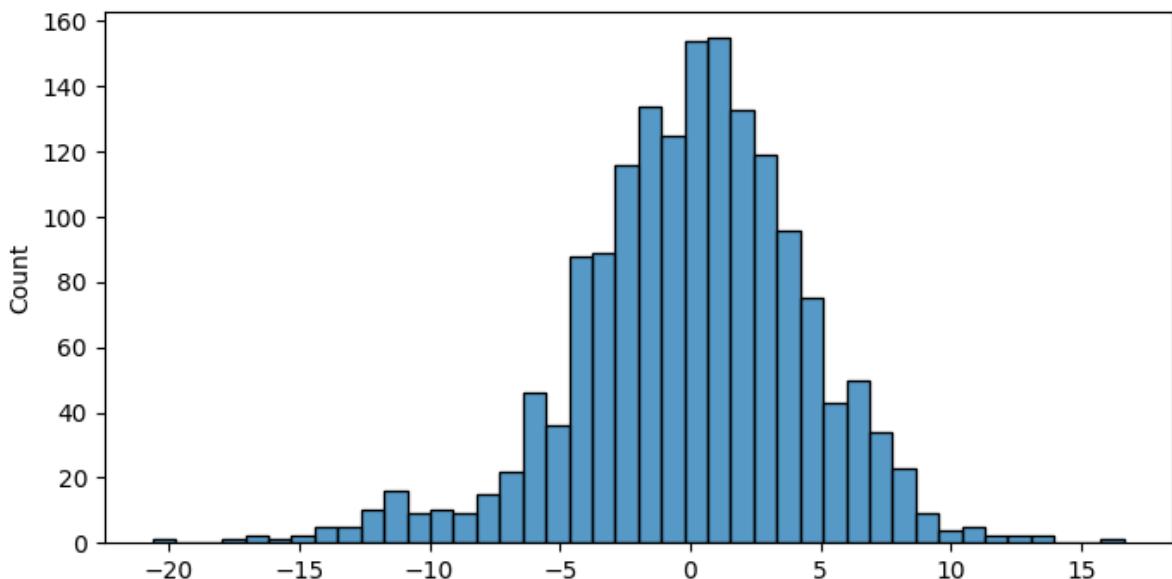
## Residuenanalyse

Damit die lineare Regression Sinn macht, sollten die Residuen normalverteilt sein. Residuen sind die Differenzen zwischen den tatsächlichen beobachteten Werten und den von einem Modell vorhergesagten Werten.

```
In [14]: residuals = model.resid

# Plotting the residuals distribution
plt.subplots(figsize=(8, 4))
plt.title('Distribution of Residuals', fontsize=18)
sns.histplot(residuals)
plt.show()
```

## Distribution of Residuals



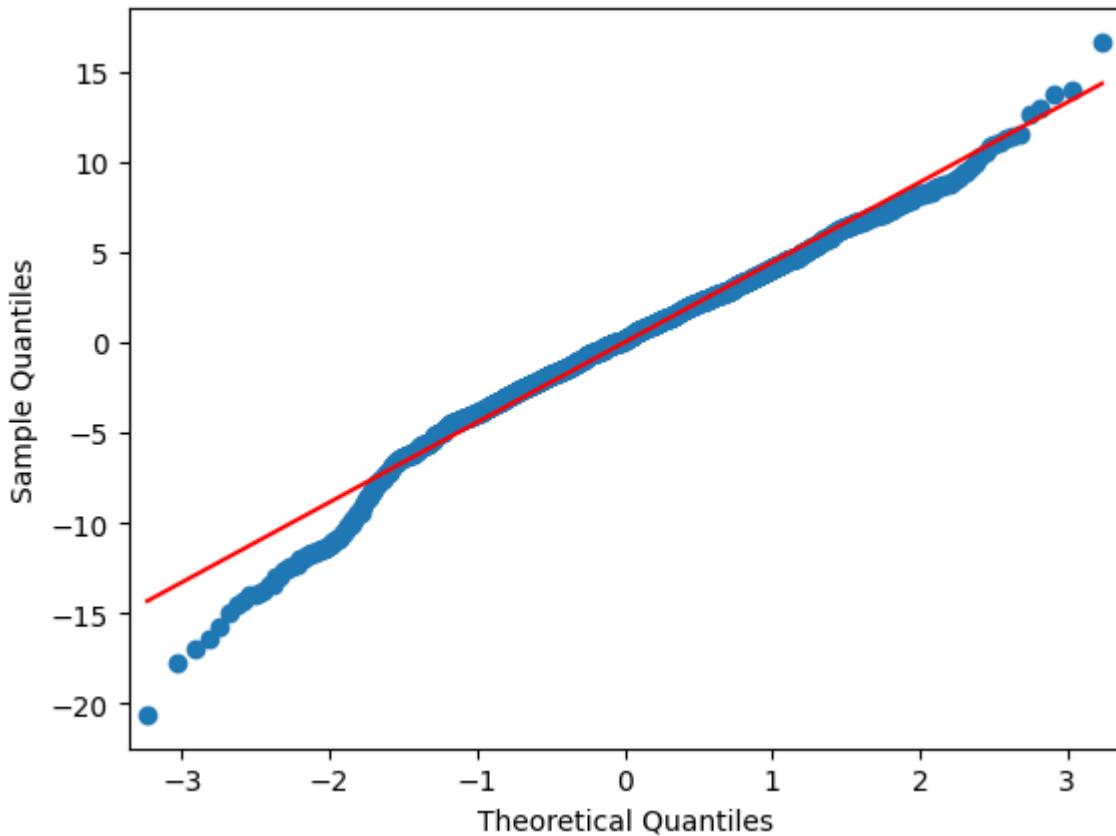
Die Residuen sind schön in der Gauss'schen Glockenkurve verteilt und somit normalverteilt. Wenn die Residuen nicht normalverteilt wären, kann dies darauf hindeuten, dass das Modell nicht die tatsächliche Datenstruktur abbildet, und die statistischen Verfahren könnten unzuverlässig sein. Das ist hier nicht der Fall.

## QQ-Plot

Zur Sicherheit machen wir noch einen QQ-Plot der Residuen.

```
In [15]: # Create a QQ plot for the residuals
qqplot(residuals, line='s')

# Display the plot
plt.show()
```



Der QQ-Plot vergleicht die Residuen mit den Quantilen der theoretischen Normalverteilung. Wenn die Residuen normalverteilt sind, sollten die Punkte wie im Diagramm oben ersichtlich im QQ-Plot entlang einer diagonalen Linie liegen. Das bedeutet, dass die Residuen mit den erwarteten Quantilen der Normalverteilung übereinstimmen. Das Modell multiple lineare Modell erklärt also die Daten gut und die Annahme der Normalverteilung für die Residuen ist gerechtfertigt.

## Logistische Regression

Eine logistische Regression ist ein statistisches Verfahren, um die Wahrscheinlichkeit eines Ereignisses vorherzusagen. Es wird eingesetzt, um binäre (ja/nein bzw. 0/1) Ergebnisse vorherzusagen.

Die logistische Regression verwendet eine logistische Funktion, um die Wahrscheinlichkeit des Ereignisses in Abhängigkeit von den unabhängigen Variablen zu modellieren. Das Ergebnis der logistischen Regression ist also ein Modell, das anzeigt, wie stark jede unabhängige Variable mit der Wahrscheinlichkeit des Ereignisses zusammenhängt.

Um eine logistische Regression hier anwenden zu können, muss die abhängige Variable also erst transformiert werden, sodass diese aus 0 und 1 Werte besteht. Dazu kann man z.B. eine neue Spalte definieren, die angibt, ob die Lebenserwartung grösser oder gleich 65 Jahre ist. Falls ja, wird eine 1 codiert, falls nein, wird eine 0 codiert.

```
In [16]: # Copy the input data set
df_input_data_log = df_input_data.copy()

# Calculate a life_exp column with 0 or 1 value for a logistic regression
df_input_data_log["life_exp_65_or_higher"] = (df_input_data_log["life_exp"])

# Define the predictor variables (features)
X = df_input_data_log[['population',
                       'gdp',
                       'bmi',
                       'polio',
                       'diphth',
                       'hiv',
                       'alcohol',
                       'measles',
                       'hep_b',
                       'inft_death',
                       'adlt_mort',
                       'thinness_teenager',
                       'thinness_children',
                       'total_exp']]

# Add a constant term to the predictor variables to represent the intercept
X = sm.add_constant(X)

# Define the response variable (target)
y = df_input_data_log['life_exp_65_or_higher']

# Create and fit the logistic regression model
model = sm.Logit(y, X, missing = 'drop')
result = model.fit()

# Print the summary of the model
print(result.summary())
```

Optimization terminated successfully.  
 Current function value: 0.178085  
 Iterations 10

### Logit Regression Results

```
=====
=====
Dep. Variable: life_exp_65_or_higher No. Observations: 1649
Model: Logit Df Residuals: 1634
Method: MLE Df Model: 14
Date: Sat, 17 Jun 2023 Pseudo R-squ.: 0.6927
Time: 23:11:31 Log-Likelihood: -293.66
converged: True LL-Null: 955.54
Covariance Type: nonrobust LLR p-value: 5e-274
=====
=====
```

	coef	std err	z	P> z	[0.025
0.975]					
const	0.8545	0.647	1.320	0.187	-0.414
2.123					
population	-4.513e-10	1.37e-09	-0.330	0.742	-3.13e-09
2.23e-09					
gdp	0.0006	9.31e-05	6.876	0.000	0.000
0.001					
bmi	0.0276	0.008	3.461	0.001	0.012
0.043					
polio	0.0073	0.005	1.348	0.178	-0.003
0.018					
diphth	0.0118	0.006	1.855	0.064	-0.001
0.024					
hiv	-1.2629	0.128	-9.855	0.000	-1.514
-1.012					
alcohol	0.1948	0.049	4.017	0.000	0.100
0.290					
measles	1.147e-05	1.43e-05	0.801	0.423	-1.66e-05
3.95e-05					
hep_b	0.0017	0.005	0.315	0.753	-0.009
0.012					
inft_death	-0.0009	0.001	-0.849	0.396	-0.003
0.001					
adlt_mort	-0.0130	0.001	-9.087	0.000	-0.016
-0.010					
thinness_teenager	-0.0279	0.038	-0.742	0.458	-0.102
0.046					
thinness_children	0.0251	0.038	0.656	0.512	-0.050
0.100					
total_exp	-0.0259	0.056	-0.464	0.643	-0.135

```
0.083
```

```
=====
=====
```

Possibly complete quasi-separation: A fraction 0.21 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

Die Konstante ist in diesem Fall 0.8545. Mit einem Signifikanzniveau von 0.05 sind also ausschliesslich die Variablen gdp, bmi, hiv, alcohol und adlt\_mort signifikant (< 0.05).

Das Pseudo-Bestimmtheitsmass beträgt 69.27%. Das bedeutet, dass 69.27% der Variabilität der abhängigen Variable durch das logistische Modell erklärt werden kann.

## Principal Component Analysis

Mit der PCA ("Principal Component Analysis" oder auch Hauptkomponentenanalyse) versucht man in den Daten eine Dimensionsreduktion zu machen. Dabei werden einzelne Variablen zusammen genommen zu einer neuen Variable, die möglichst eine maximale Varianz in den Daten erklären.

```
In [48]: # Copy the input data set
df_input_data_pca = df_input_data.copy()

# drop non numeric columns
df_input_data_pca = df_input_data_pca.drop('country', axis=1)
df_input_data_pca = df_input_data_pca.drop('status', axis=1)

# drop NaN values
df_input_data_pca.dropna(inplace=True)
```

Nun skalieren wir alle numerischen Spalten mit dem StandardScaler.

```
In [49]: # Scale the input features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_input_data_pca)
```

Wir versuchen eine Hauptkomponentenanalyse mit 2 Komponenten.

```
In [50]: # Keep 2 components
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_scaled)
```

Nun kann das Resultat analysiert werden.

```
In [51]: # Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
print("Explained variance ratio:", explained_variance_ratio)
```

```
Explained variance ratio: [0.3067837  0.14494487]
```

Die "explained variance ratio" gibt den Anteil, den der Faktor an der Gesamtvarianz der beobachteten Variablen erklärt. Das bedeutet, dass mit diesem Resultatvektor [0.3067837 0.14494487] 30.68% der Varianz durch Komponente 1 erklärt ist und 14.50 durch Komponente 2.

## Hypothesentests

**Fragestellung: Sind die Bruttoinlandprodukte (BIP) über die verschiedenen Länder normalverteilt?**

Um die Frage zu beantworten, kann mit einem Shapiro-Wilk-Test auf Normalverteilung geprüft werden.

H<sub>0</sub> (Nullhypothese): Die Bruttoinlandprodukte sind normalverteilt

H<sub>1</sub> (Arbeitshypothese): Die Bruttoinlandprodukte sind nicht normalverteilt

Als Signifikanzniveau wählen wir 5%.

```
In [22]: # Copy the input data set
df_hypo_data_log = df_input_data.copy()

# Shapiro-Wilk-Test to check if data is normally distributed
statistic, p_value = shapiro(df_hypo_data_log['gdp'])
```

Wenn der p-Wert kleiner oder gleich wie das Signifikanzniveau ist, wird die Nullhypothese verworfen.

```
In [23]: # significance level
alpha = 0.05

if p_value > alpha:
    print("Die Nullhypothese (Daten sind normalverteilt) kann nicht verworfen")
else:
    print("Die Nullhypothese (Daten sind normalverteilt) wird verworfen.")

Die Nullhypothese (Daten sind normalverteilt) kann nicht verworfen werden.
```

```
In [24]: p_value
```

```
Out[24]: 1.0
```

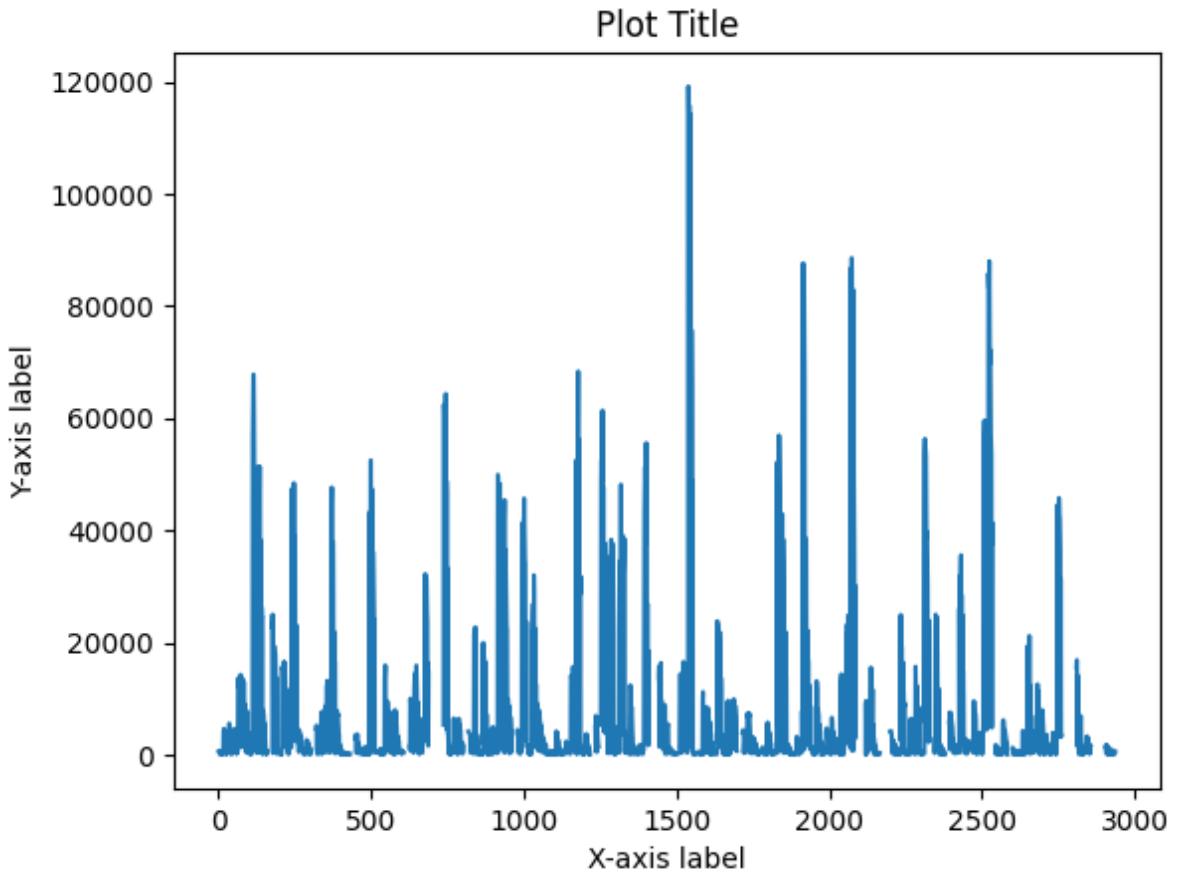
Der p-Wert beträgt 1.0, was hier bedeutet, dass die Nullhypothese nicht verworfen werden kann und die Daten somit mit einer Wahrscheinlichkeit von 95% normalverteilt sind. Ein so hoher Wert ist allerdings erstaunlich und es ist fachlich schwer vorstellbar, dass die Daten normalverteilt sein können. Am Besten visualisieren wir mal das Datenset.

```
In [25]: plt.plot(df_hypo_data_log['gdp'])
```

```

plt.xlabel('X-axis label') # Set the label for the x-axis
plt.ylabel('Y-axis label') # Set the label for the y-axis
plt.title('Plot Title')   # Set the title of the plot
plt.show()

```



Auf einen Blick sieht man, dass die Daten keiner schönen Normalverteilung (Glockenkurve) folgen. Ein gutes Beispiel dafür, dass man sich in der Statistik nie auf einen einzelnen Test verlassen sollte. Am Besten prüfen wir die Normalverteilung noch mit einem anderen Test.

## Einstichprobenverfahren

Die Fragestellung von vorhin können wir nun mit einem Einstichprobenverfahren versuchen zu beantworten. Es soll ein Kolmogorow-Smirnow-Test durchgeführt werden.

```

In [26]: # Copy the input data set
df_hypo_data_log = df_input_data.copy()

# remove na values
df_hypo_data_log['gdp'].dropna()

# Nullhypothesis: The gdp data is normally distributed
null_distribution = 'norm'

# Kolmogorow-Smirnow-Test
statistic, p_value = kstest(df_hypo_data_log['gdp'], null_distribution)

```

```
# significance level
alpha = 0.05
if p_value > alpha:
    print("Die Nullhypothese (Daten sind normalverteilt) kann nicht verworfen")
else:
    print("Die Nullhypothese (Daten sind normalverteilt) wird verworfen.")
```

Die Nullhypothese (Daten sind normalverteilt) wird verworfen.

Die Bruttoinlandprodukt-Daten sind also *nicht* normalverteilt. Der p-Wert ist kleiner als das Signifikanzniveau.

## Zeitreihenanalyse

Leider ist das Datenset nicht wirklich geeignet für eine Zeitreihenanalyse. Es gibt einfach zu wenige Messwerte für ein einzelnes Land. Um die Methodik trotzdem durchspielen zu können, soll an dieser Stelle aber einfach auf die Werte der Schweiz gefiltert werden.

```
In [27]: # Copy the input data set
df_input_data_ts = df_input_data.copy()

# Filter for Switzerland's values
df_input_data_ts = df_input_data_ts.query("country == 'Switzerland'")

df_input_data_ts
```

Out[27]:

	country	year	status	life_exp	adlt_mort	inft_death	alcohol	perc_exp
2521	Switzerland	2015	Developed	83.4	49.0	0	NaN	0.000000
2522	Switzerland	2014	Developed	83.2	51.0	0	9.61	19479.911610
2523	Switzerland	2013	Developed	83.0	52.0	0	9.73	19099.045060
2524	Switzerland	2012	Developed	82.7	54.0	0	9.86	18379.329740
2525	Switzerland	2011	Developed	82.6	55.0	0	9.99	18822.867320
2526	Switzerland	2010	Developed	82.3	57.0	0	10.01	2198.590865
2527	Switzerland	2009	Developed	82.1	6.0	0	10.15	14714.825880
2528	Switzerland	2008	Developed	82.0	6.0	0	10.29	2084.255535
2529	Switzerland	2007	Developed	81.7	63.0	0	10.44	11892.334290
2530	Switzerland	2006	Developed	81.5	65.0	0	10.24	10598.081870
2531	Switzerland	2005	Developed	81.1	66.0	0	10.15	10055.349810
2532	Switzerland	2004	Developed	81.0	69.0	0	10.55	9495.540576
2533	Switzerland	2003	Developed	85.0	72.0	0	10.82	842.276809
2534	Switzerland	2002	Developed	84.0	74.0	0	10.85	6853.628494
2535	Switzerland	2001	Developed	82.0	75.0	0	11.12	6478.346135
2536	Switzerland	2000	Developed	79.7	78.0	0	11.26	5834.582046

16 rows × 22 columns

Es ist also eine sehr bescheidene Zeitreihe mit nur 16 Zeilen (Jahr 2000-2015).

In [28]:

```
# Filter for relevant columns
df_input_data_ts = df_input_data_ts[['year', 'gdp']]

# Convert 'year' column to datetime format
df_input_data_ts['year'] = pd.to_datetime(df_input_data_ts['year'], format='%Y')

# Set 'year' column as the time index
df_input_data_ts.set_index('year', inplace=True)

# Sort the DataFrame based on the date index
df_input_data_ts = df_input_data_ts.sort_index()

df_input_data_ts
```

```
/var/folders/_s/gwlgn2gs5ts1bjthvw7gxllrr0000gn/T/ipykernel_42091/292602517
1.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df_input_data_ts['year'] = pd.to_datetime(df_input_data_ts['year'], format='%Y')
```

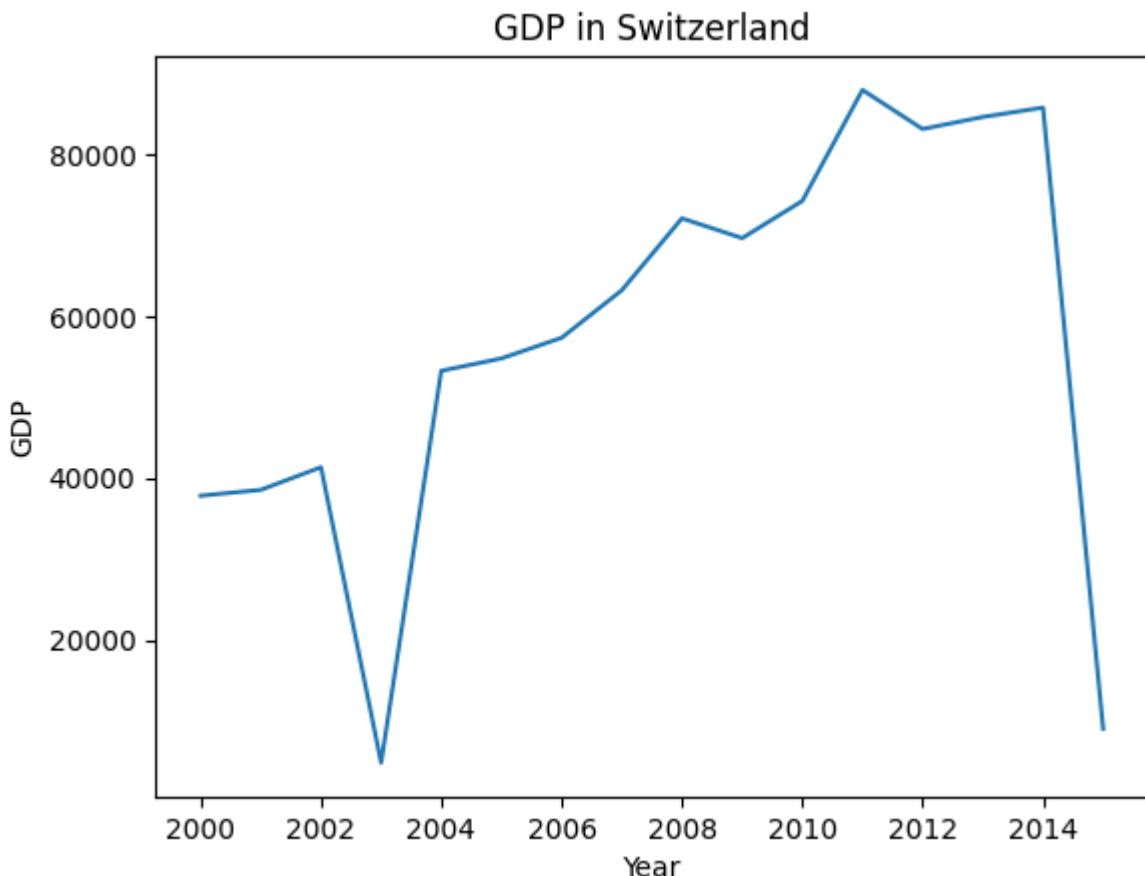
Out[28]:

gdp

year	gdp
2000-01-01	37813.23426
2001-01-01	38538.64447
2002-01-01	41336.72192
2003-01-01	4796.56497
2004-01-01	53255.97631
2005-01-01	54797.54663
2006-01-01	57348.92788
2007-01-01	63223.46778
2008-01-01	72119.56870
2009-01-01	69672.47100
2010-01-01	74276.71842
2011-01-01	87998.44468
2012-01-01	83164.38795
2013-01-01	84658.88768
2014-01-01	85814.58857
2015-01-01	8989.84240

In [29]:

```
# Plot the GDP data
plt.plot(df_input_data_ts.index, df_input_data_ts['gdp'])
plt.xlabel('Year')
plt.ylabel('GDP')
plt.title('GDP in Switzerland')
plt.show()
```



```
In [30]: # Perform Augmented Dickey–Fuller test for stationarity
result = sm.tsa.stattools.adfuller(df_input_data_ts['gdp'])
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

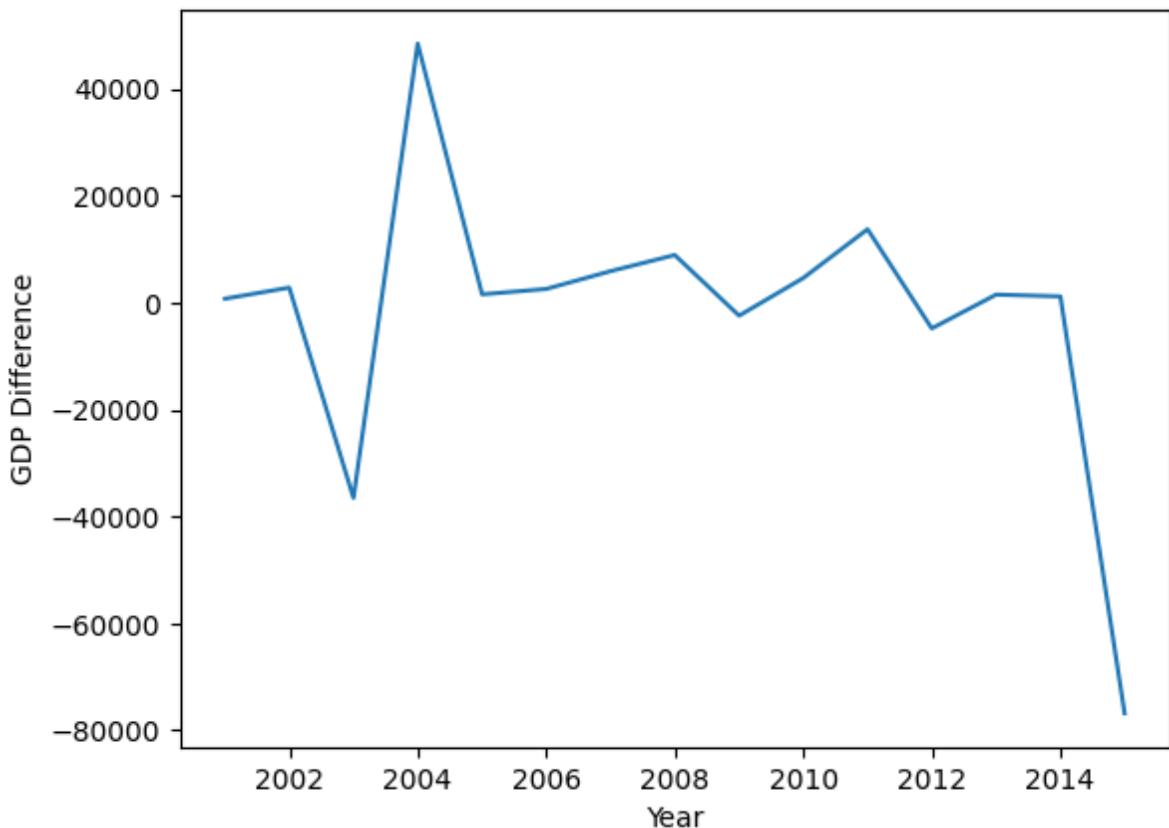
ADF Statistic: -1.8383634292707938  
p-value: 0.361588423100048

Der p-Wert ist signifikant höher als 0.05, was auf eine nicht-stationäre Zeitreihe hinweist.  
Wir wollen nun eine erste Differenzierung durchführen. Durch die Berechnung der Differenz werden die absoluten Werte der Zeitreihe entfernt und es bleiben nur noch die Veränderungen zwischen den Beobachtungen übrig. Dadurch kann man saisonale Muster oder Trends eliminieren.

```
In [31]: # Take the first difference of the GDP data
df_input_data_ts['gdp_diff'] = df_input_data_ts['gdp'].diff()
df_input_data_ts['gdp_diff'].dropna(inplace=True)

# Plot the differenced data
plt.plot(df_input_data_ts.index[:,], df_input_data_ts['gdp_diff'])
plt.xlabel('Year')
plt.ylabel('GDP Difference')
plt.title('Differenced GDP Data for Switzerland')
plt.show()
```

## Differenced GDP Data for Switzerland



Nun prüfen wir nochmals die Stationarität aber auf der ersten Differenzierung (Spalte gdp\_diff).

```
In [32]: # Perform Augmented Dickey-Fuller test for stationarity
df_input_data_ts.dropna(inplace=True)
result = sm.tsa.stattools.adfuller(df_input_data_ts['gdp_diff'])
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

```
ADF Statistic: -3.2856189953249832
p-value: 0.015533672227103848
```

Die Stationarität konnte durch die Differenzierung also erfolgreich sichergestellt werden.  
Der p-wert ist nun signifikant kleiner als 0.05.

Um später eine ARIMA "Autoregressive Integrated Moving Average" Vorhersage zu machen, wollen wir die optimalen Werte für p (wieviele vorhergehende Schritte sollen verwendet werden um den nächsten vorherzusagen), d (wie oft soll differenziert werden) und q (wieviele Fehlerinformationen sollen verwendet werden). Dazu kann man alle möglichen Kombinationen von p, d und q in einem gewissen Range testen um die optimale Konfiguration zu finden.

```
In [33]: # Explicitly specify the frequency as 'A' for annual data
df_input_data_ts.index = pd.date_range(start=df_input_data_ts.index[0], peri
# Define the range of values to search for p, d, and q
```

```

p_values = range(1, 3) # Range for AR parameter
d_values = range(0, 2) # Range for differencing parameter
q_values = range(0, 3) # Range for MA parameter

best_aic = np.inf
best_order = None

# Iterate over all possible combinations of p, d, and q
for p in p_values:
    for d in d_values:
        for q in q_values:
            order = (p, d, q)
            try:
                # Fit the ARIMA model
                model = sm.tsa.ARIMA(df_input_data_ts['gdp_diff'], order=order)
                model_fit = model.fit()

                # Check if the current model has a lower AIC value
                if model_fit.aic < best_aic:
                    best_aic = model_fit.aic
                    best_order = order
            except:
                continue

# Print the best order
print("Best (p, d, q):", best_order)

```

Best (p, d, q): (2, 1, 0)

```

/Users/pathir/Desktop/FFHS – Data Science/3 – CAS Statistische Datenanalyse/StatDa/Semesterarbeit/statda_py/lib/python3.9/site-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters found. Using zeros as starting parameters.
    warn('Non-stationary starting autoregressive parameters')
/Users/pathir/Desktop/FFHS – Data Science/3 – CAS Statistische Datenanalyse/StatDa/Semesterarbeit/statda_py/lib/python3.9/site-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using zeros as starting parameters.
    warn('Non-invertible starting MA parameters found.')
/Users/pathir/Desktop/FFHS – Data Science/3 – CAS Statistische Datenanalyse/StatDa/Semesterarbeit/statda_py/lib/python3.9/site-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters found. Using zeros as starting parameters.
    warn('Non-stationary starting autoregressive parameters')
/Users/pathir/Desktop/FFHS – Data Science/3 – CAS Statistische Datenanalyse/StatDa/Semesterarbeit/statda_py/lib/python3.9/site-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using zeros as starting parameters.
    warn('Non-invertible starting MA parameters found.')
/Users/pathir/Desktop/FFHS – Data Science/3 – CAS Statistische Datenanalyse/StatDa/Semesterarbeit/statda_py/lib/python3.9/site-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters found. Using zeros as starting parameters.
    warn('Non-stationary starting autoregressive parameters')

```

Nun nutzen wir die oben ermittelte Optimalkonfiguration von p, d und q (2,1,0), um das ARIMA Modell zu fitten.

```
In [34]: # Determine the optimal p, d, and q values using grid search or other method
p = 2
d = 1
q = 0

# Create and fit the ARIMA model
model = sm.tsa.ARIMA(df_input_data_ts['gdp_diff'], order=(p, d, q))
model_fit = model.fit()

# Print the model summary
print(model_fit.summary())
```

## SARIMAX Results

```
=====
===
Dep. Variable:          gdp_diff    No. Observations:      15
Model:                 ARIMA(2, 1, 0)    Log Likelihood:   -162.
868
Date:                 Sat, 17 Jun 2023    AIC:                  331.
737
Time:                 23:11:37        BIC:                  333.
654
Sample:                12-31-2001    HQIC:                  331.
559
                           - 12-31-2015

Covariance Type:     opg
=====
```

	coef	std err	z	P> z	[0.025	0.9
75]						
---						
ar.L1	-1.0198	0.331	-3.083	0.002	-1.668	-0.
371						
ar.L2	-0.5289	0.562	-0.941	0.347	-1.631	0.
573						
sigma2	7.329e+08	8.31e-10	8.82e+17	0.000	7.33e+08	7.33e
+08						
=====						
=====						
Ljung-Box (L1) (Q):			0.13	Jarque-Bera (JB):		
3.22						
Prob(Q):			0.72	Prob(JB):		
0.20						
Heteroskedasticity (H):			1.52	Skew:		
-0.83						
Prob(H) (two-sided):			0.66	Kurtosis:		
4.67						
=====						
=====						

### Warnings:

- [1] Covariance matrix calculated using the outer product of gradients (complex-step).
- [2] Covariance matrix is singular or near-singular, with condition number 3.79e+34. Standard errors may be unstable.

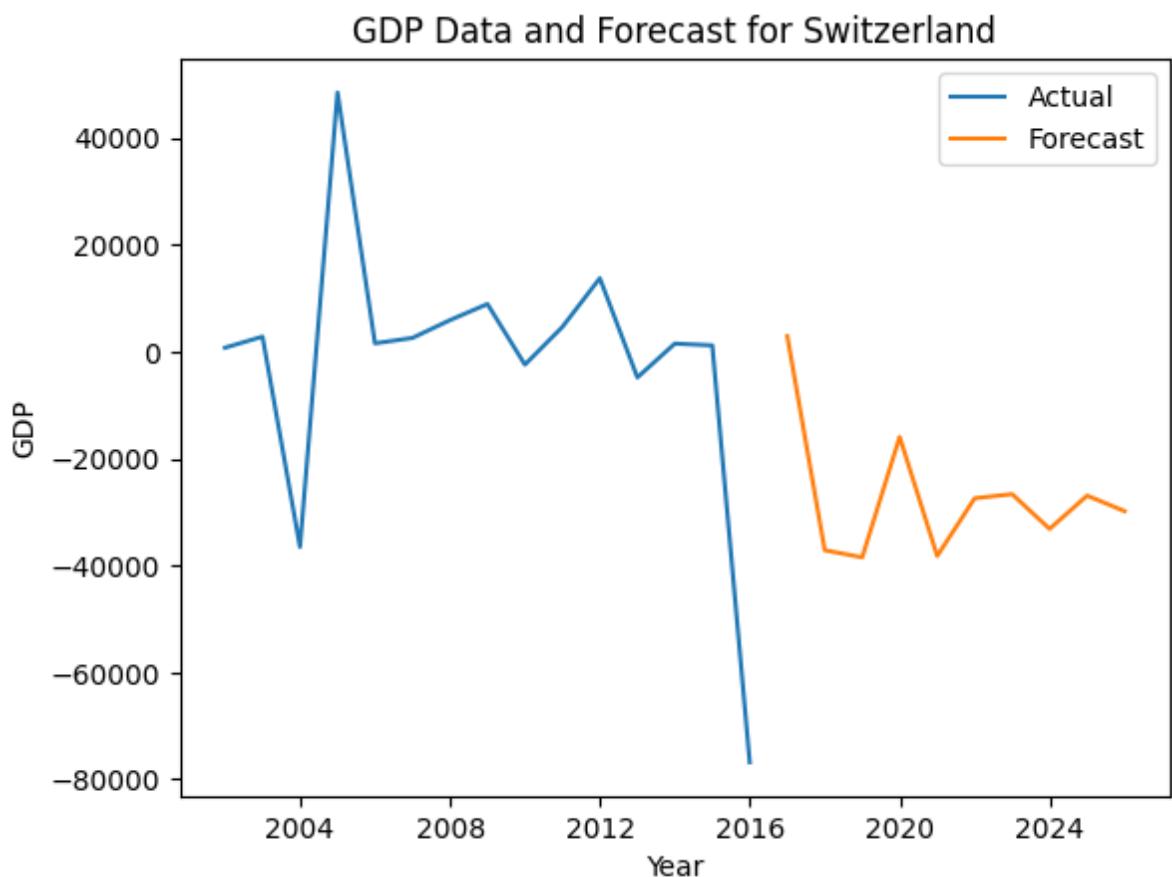
Wenn wir mit dem Modell nun die nächsten 10 Jahre vorhersagen wollen, sieht das folgendermassen aus:

```
In [35]: # Forecast future values
forecast = model_fit.forecast(steps=10)
```

```

# Plot the original data and the forecasted values
plt.plot(df_input_data_ts.index, df_input_data_ts['gdp_diff'], label='Actual')
plt.plot(forecast.index, forecast, label='Forecast')
plt.xlabel('Year')
plt.ylabel('GDP')
plt.title('GDP Data and Forecast for Switzerland')
plt.legend()
plt.show()

```



In blau sind die effektiven Werte aus dem Datenset abgebildet, in gelb die vorhergesagten darauffolgenden 10 Jahre. Da dies ein altes Datenset ist und die effektiven Werte für die Jahre nach 2015 sogar auffindbar wären im Internet, könnte man diese nun mit der Vorhersage verglichen werden. Auf diesen Aufwand wird an dieser Stelle aber verzichtet. Es ging hier vor Allem um die Methodik der Zeitreihenanalyse.